PARS - Multiscreen Web App Platform

- A Position Paper for the Fourth W3C Web and TV workshop -

Hyojin Song **Soonbo Han** **Dong-Young Lee**

LG Electronics

{hyojin22.song, soonbo.han, dongyoung.lee}@lge.com

Abstract

As more and more devices around us are getting connected or "smart", there are growing needs for those devices to provide coordinated or "orchestrated "[1] user experiences. In this paper we present Pars[2], a multiscreen web application platform, to enable such user experiences. A Pars web app consists of components that can run distributed on a set of devices as if they are running on a single device. Each component can migrate or be replicated to another device in the middle of execution, and synchronizes its states with other components on the same or other devices. The Pars platform consists of two layers, the Pars framework including a daemon and JavaScript library, and an underlying networking layer.

1 Introduction

According to current predictions, the number of connected devices will reach 30-50 billion in 2020, which is 4 to 6 times more than the estimated world population at that time[3]. Using more than one device simultaneously is becoming more and more common. For example, today, 77% of the time when we're watching a TV, we're using another device[4]. Instead of relying on the user to coordinate usage of all those devices, it would be preferable for the devices to provide an orchestrated user experience.

There may be several approaches to realize such an orchestration. In one approach, an application on one (master) device acts as a coordinator and controls the other (slave) devices through their exposed APIs. In this approach, the services provided by the APIs are usually fixed, limiting the experience that it can implement. Furthermore, the UI shown on a slave device is pre-defined by the slave device rather than the application on the master device, failing to provide a unified and refined user experience.

In another approach, an application developer pre-defines the role of each device and writes device-specific application for each device. Although this approach overcomes the limitation of fixed services and UI of the first approach, it still requires that the role of each device is pre-defined. Furthermore, the applications must be installed in all devices prior to execution, making an obstacle to deployment. DIAL[5] protocol is a solution for the pre-installation requirement.

In the last approach, the role of each device does not need to be pre-defined. An application is defined as a set of components, and each component can migrate or be duplicated among devices even in the middle of execution,

[1] This word was coined in http://www.bbc.co.uk/blogs/researchanddevelopment/2011/02/orchestrated-media---beyond-se-1.shtml.

[2] **Pars** is a Latin word meaning part, portion, or share.

[3] *Envisioning emerging technology for 2012 and beyond*, Envisioning Tech., http://envisioningtech.com/envisioning2012.

[4] http://www.google.com/think/research-studies/the-new-multi-screen-world-study.html.

[5] http://www.dial-multiscreen.org.

providing the highest flexibility in usage scenarios and the most convenience in development. The open web platform is an ideal basis for implementing a multiscreen application. First of all, it is a cross platform that is supported by virtually all smart devices. Secondly, its sandbox ensures an alien application code to be executed safely without an installation or pre-examination process.

Based on these observations, we are exploring a web-based multiscreen application platform, named *Pars*, adopting the last approach above. A Pars web app consists of components that can run distributed on a set of devices as if they are running on a single device. Each component can migrate or be replicated to another device in the middle of execution, and synchronizes its states with other components on the same or other devices.

2 Design Overview

The following figure shows service scenarios and the architecture of the Pars platform.

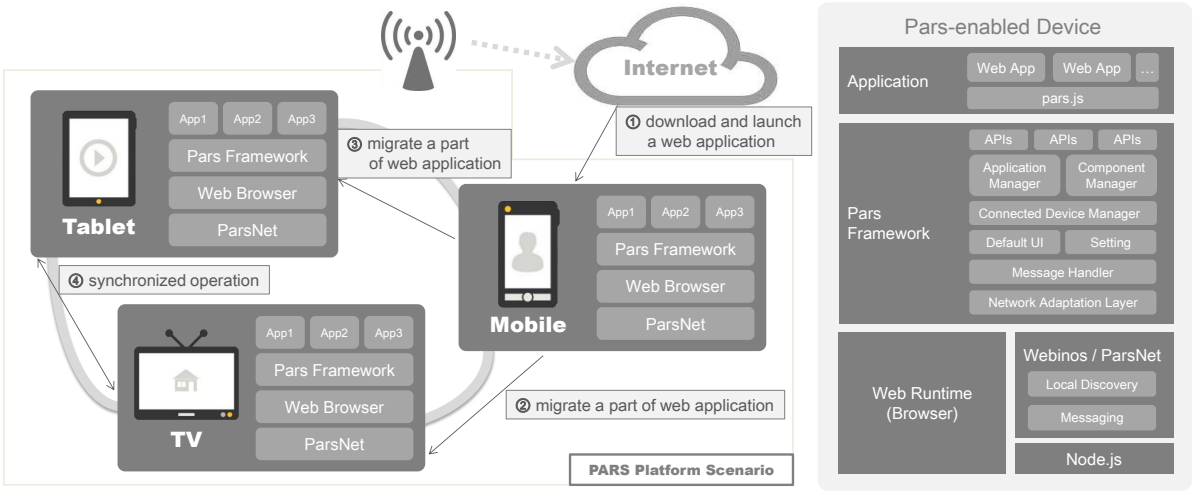

Figure 1: Scenario and Architectural Overview how the Pars platform is embedded in a device

The scenario depicted in Figure 1 shows the lifecycle of a Pars-enabled web app on the Pars platform. A user downloads and launches a web app to use in a mobile device. While using it, the user can run parts of it on another device. For example, a user can start using a Pars-enabled video-on-demand web application on his smartphone, and then send the video player of the web app to a smart TV, leaving lists and comments parts on the phone. The user can move or replicate any parts to other devices. For another example, a user can have a smart TV display a map, while navigating a search result list on her phone. Whenever she selects an item in the list on the phone, the map on the TV moves to show the selected item. It is also possible for her to have the list shown both on the phone and on the TV.

In our current design, a component is a basic unit composing a Pars-enabled web app, which can be moved or replicated to another device. A component includes a JavaScript object to define actions and store state information, and HTML code for rendering. They are serialized and transferred to another device when the component is moved or replicated.

To receive a migrating or replicated component and state information, the Pars platform runs a *daemon*. The daemon is a service web app running on the web runtime (e.g., a browser). It can be embedded in the web runtime or exist as a web app launched by the user to start the Pars platform. It launches a new Pars web app with delivered

components or inserts the component into an existing Pars web app, in case it already exists.

The Pars platform consists of two layers: the Pars framework including the daemon, and an underlying networking layer. The Pars framework provides multiscreen-related APIs for developers, and manages connected devices where Pars web apps can run. The underlying networking layer provides local discovery and messaging services. The following section describes details of the two layers.

3 Technical Details

3.1 Pars Framework Layer

The Pars framework is a JavaScript library designed to build a Pars-enabled web app. A Pars-enabled web app should be developed using this framework as a set of components. A component, composed of variables (or properties) and functions (or methods), is the basic unit of a Pars-enabled web app. It can be moved or replicated to, and pulled from other devices. The framework provides APIs to handle function calls between components, manage states of a component, and support user interactions. Developers can use them simply by embedding pars.js in the web app.

Interactions with other components, whether on the same device or another, are done by a remote function call. In case a component is replicated, it is possible to designate whether the function is executed at all replicas or not. For example, a function for a money-transfer transaction should be executed only once whereas a function to show the updated balance should be executed on all devices. When a remote function call is requested, a callback can be optionally registered. After the requested function is executed, the callback will be called with its return value as parameter.

A component has state information including current values of its variables. Although it might be convenient to transfer all the state information automatically, often times much of the state information is useless or even confusing on another device, because of e.g., adaptation to the host device. In our current design, each component implements functions to save and restore states, which are invoked by the platform before and after every migration or replication.

The UI to manipulate component migration and replication can be provided by the platform, or custom-defined by a Pars web app. Figure 2 shows the default UI provided by the platform. A user can manipulate the components just by clicking buttons for migration, replication, pull, and remove; and dragging components to a device icon. The sync checkbox is used to indicate whether the states of the components in the original and the target device should be synchronized. For example, if all components are replicated to another device with the sync unchecked, the target device would have the same web app with an initial state, which is the same as newly launching the app on the target device.

Alternatively, the UI can be custom-defined by a Pars web app, e.g., to provide simpler interactions. The Pars framework provides APIs that are necessary to implement the UI.

3.2 Networking Layer

The Pars framework works on a network adaptation layer, enabling it to run any underlying networking protocols that provide local discovery and messaging services. One example of such protocols is webinos[6], on which our first implementation was based.

[6] http://www.webinos.org.

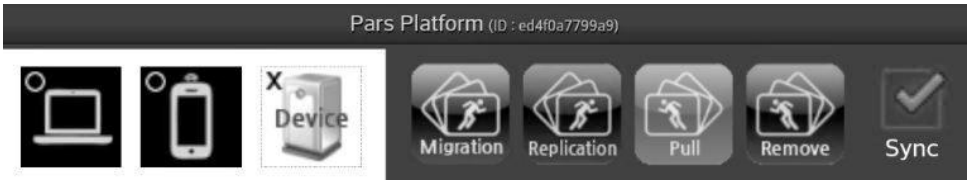

Figure 2: The user can interact with the default Pars UI provided by the Pars Framework to send or receive components

The network adaptation layer requires three function be implemented: `discovery()`, `registerMessageHandler()`, and `send()`. The `discovery()` function requests to start discovery and registers a callback that will be called when the information of discovered devices is updated. The `registerMessageHandler()` registers a callback to handle a received message. The `send()` function sends a message to target devices.

For systems where webinos is not available or stable enough, we implemented a simple communications protocol based on Socket.IO[7] and node.js[8], named *ParsNet*. Since Socket.IO is a WebSocket-like library, it can be replaced with the WebSocket APIs if necessary.

ParsNet discovers another device that runs ParsNet within a local network by sending a broadcast message. Once a device is discovered, the Pars platform puts it on the list of target devices. Then, any component of a Pars web app on the device can send and receive messages between the devices.

Other technologies can be adopted to support the underlying communications for the Pars platform. For example, the NSD (Network Service Discovery)[9] seems to be a good candidate for discovery, and the WebSocket, WebRTC, or postMessage could be adopted or extended to provide messaging. With the help of those ongoing web technology developments, the Pars platform would run on a web browser in the near future.

4 Related Work

Since a web page has underlying tree structure, there were previous attempts to implement a web-based distributed application platform utilizing this property, e.g. by KAIST[10] and NTT[11]. DIAL and the presentation API[12] also provide part of the Pars platform features.

5 Conclusion

We have presented Pars, a multiscreen web application platform. Although the design and implementation of Pars platform is still immature, leaving research opportunities in security, for example, it proves the feasibility of a web-based multiscreen application platform.

[7] http://socket.io/.
[8] http://nodejs.org/.
[9] http://www.w3.org/TR/discovery-api/.
[10] http://www.w3.org/2011/09/webtv/papers/W3C_3rd_WebTV_position_paper_KAIST_Final_submit.pdf.
[11] "*Cross-screen Web Applications*", Dr. Kiyoshi Tanaka, 3rd FOKUS Media Web Symposium.
[12] http://webscreens.github.io/presentation-api/.